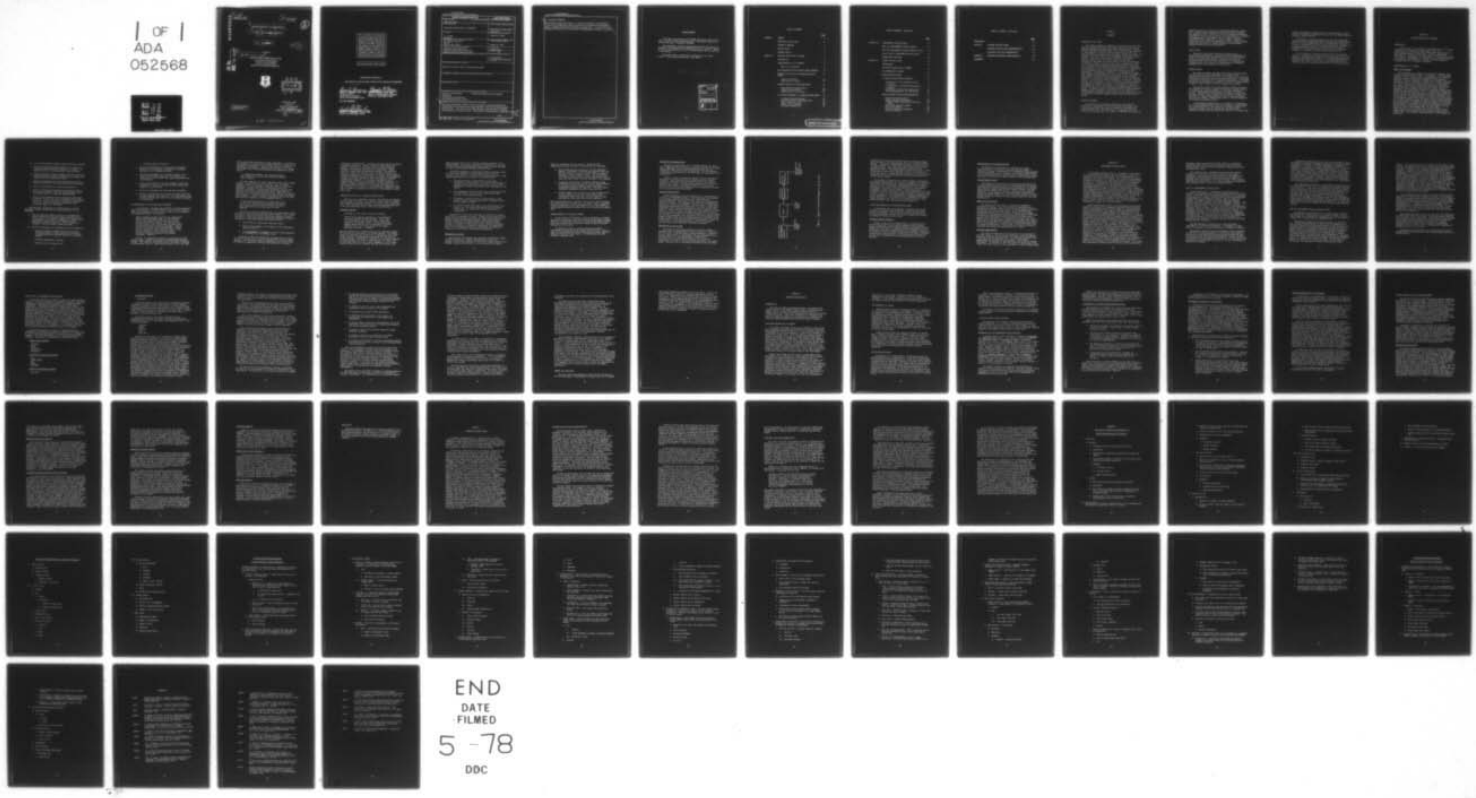
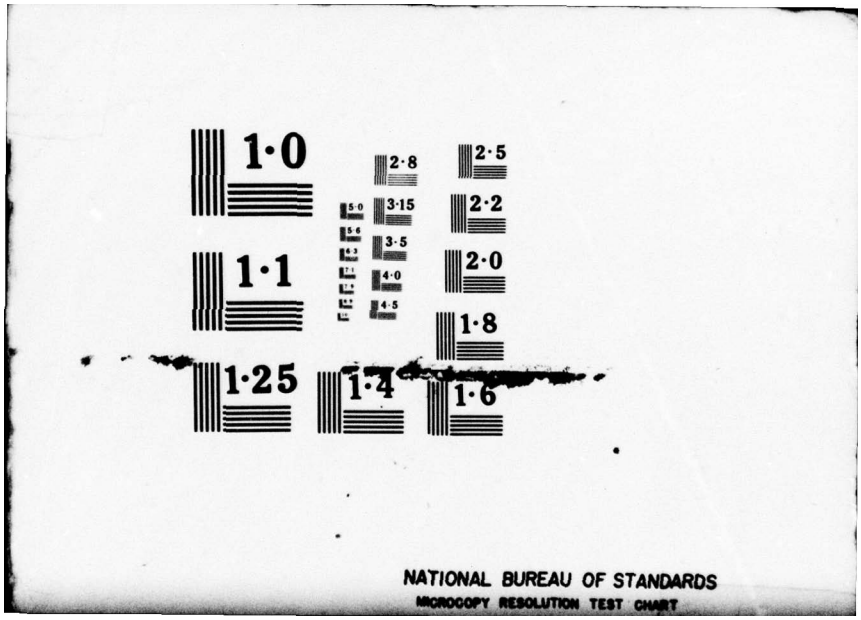
AD-A052 568 MITRE CORP BEDFORD MASS F/G 17/2
BUILDING BLOCKS FOR C3 SYSTEMS.(U)
MAR 78 J A CLAPP, M HAZLE F19628-77-C-0001
UNCLASSIFIED MTR-3504 ESD-TR-77-360 NL

1 OF 1
ADA
052568



END
DATE
FILMED
5-78
DDC


1·0
1·1
1·25
2·8
3·15
3·5
4·0
4·5
1·4
2·5
2·2
2·0
1·8
1·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

AD A052568

AD No.
DDC FILE COPY

(18) ESD-TR-77-360 (19)

(14) MTR-3504

(2)

# BUILDING BLOCKS FOR C³ SYSTEMS

(10) BY J. A. CLAPP AND M. HAZLE

(11) MARCH 1978

(12) 71 p.

(9) Technical rept.

Prepared for

DEPUTY FOR DEVELOPMENT PLANS
ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
Hanscom Air Force Base, Bedford, Massachusetts

DDC
RECEIVED
APR 11 1978
B

Project No. 7060
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract No. F19628-77-C-0001

(15)

Approved for public release;
distribution unlimited.

235 050

mt

When U.S. Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Do not return this copy. Retain or destroy.

## REVIEW AND APPROVAL

This technical report has been reviewed and is approved for publication.

DENNIS K. KAWAMURA, Capt, USAF
Project Officer
Technological Planning
Deputy for Development Plans

VAUGHN K. GRACE, LtCol, USAF
Director, Technological Planning
Deputy for Development Plans

FOR THE COMMANDER

MICHAEL H. ALEXANDER, Colonel, USAF
Deputy for Development Plans

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>ESD-TR-77-360 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE *(and Subtitle)*<br>BUILDING BLOCKS FOR $C^3$ SYSTEMS | | 5. TYPE OF REPORT & PERIOD COVERED |
| | | 6. PERFORMING ORG. REPORT NUMBER<br>MTR-3504 |
| 7. AUTHOR(s)<br>J.A. Clapp<br>M. Hazle | | 8. CONTRACT OR GRANT NUMBER(s)<br>F19628-77-C-0001 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>The MITRE Corporation<br>Box 208<br>Bedford, MA 01730 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS<br>Project No. 7060 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>Deputy for Development Plans<br>Electronic Systems Division, AFSC<br>Hanscom Air Force Base, MA 01731 | | 12. REPORT DATE<br>MARCH 1978 |
| | | 13. NUMBER OF PAGES<br>70 |
| 14. MONITORING AGENCY NAME & ADDRESS *(if different from Controlling Office)* | | 15. SECURITY CLASS. *(of this report)*<br>UNCLASSIFIED |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

DATA FLOW ANALYSIS
REUSABLE
SOFTWARE ADAPTABILITY
SOFTWARE BUILDING BLOCKS
SOFTWARE SYSTEM DEVELOPMENT

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

This report contains the results of a one-year study to assess the feasibility of constructing the information processing components of $C^3_U$ systems from reusable building blocks. The objective is to reduce the time, cost, and risk of acquiring and modifying $C^3_U$ computer systems. Three kinds of building blocks are identified:

next page

(over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

20. Abstract (continued)

requirements, design, and software. A data flow architecture is proposed as a framework for partitioning a system into functional components with flexibility to adapt to changing requirements and different configurations. A preliminary plan is presented for further work to demonstrate the building block concept for $C^3$ systems.

## ACKNOWLEDGMENTS

This report was prepared by The MITRE Corporation under Project 7060, Iterative Software Synthesis (ISS). The Project Officer for ISS has been Capt. D. Kawamura, ESD/XRE.

The man/machine interface characteristics model that appears in Appendix I was developed and documented by Nancy C. Goodwin. Donald R. Peterson performed much of the $C^3$ system review that is reflected in this report.

The authors wish to thank Jane S. McCarthy for her expert support in the typing and editing of this report.

ACCESSION for
NTIS White Section
DDC Buff Section
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A

TABLE OF CONTENTS





PRECEDING PAGE BLANK-NOT FILMED

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Concluded)

# SECTION I

# SUMMARY

## OBJECTIVE OF THIS STUDY

This report presents the results of a one-year study under the Iterative Software Synthesis (ISS) project which was a part of the MITRE Technology Base program sponsored by the Electronic Systems Division (ESD/XR) of the Air Force. The objective of this study was to investigate and evaluate technology which could provide more rapid deployment and modification of computer system components of $C^3$ systems. The current methods for development of military computer systems, consisting of both computer hardware and software, typically take at least four to six years from the time an operational requirement is defined until an operational capability is delivered. That is often too long to wait, especially to find out if the originally specified requirements were the right ones. It is necessary to put together systems more quickly. Furthermore, the operational concepts for employing automation in $C^3$ systems are still evolving based on actual experience, changing tactical requirements, and changing technology. It is therefore essential that computer systems supporting tactical applications be capable of rapid adaptation, modification, and extension to meet changing requirements. Typically, modifications might be for the following reasons: to process new kinds of messages, to accept data from new sources, such as sensors or other $C^3$ systems; to change the format and contents of displays; to add to or to restructure data bases to accommodate new data or new patterns of data usage; to improve system response time; and to extend the automated capability of the system. In today's systems, such modifications are time-consuming, expensive, and error-prone. The real cost of our inability to rapidly deploy and modify computer systems can be measured as a loss of capability to perform required mission functions when they are needed.

## TECHNICAL APPROACH

The selection of a technical approach to this study was dictated by a requirement to find a near-term (within about five years) solution which would also be amenable to exploiting new technical advances in the longer term. The approach which has evolved over the past year is to develop a framework and supporting

methods and tools for efficiently utilizing and adapting a common set of computer system components across many tactical systems. This approach relies on the notion that the information processing requirements of each tactical system can be divided into (1) mission- and system-specific requirements, and (2) common functional requirements which support most tactical systems. Examples of common functions are data base management, computer system resource scheduling and allocation, display generation and management, message processing, and report generation.

## PROJECT GOALS

The goals for the first year of this project were to investigate the feasibility of defining $C^3$ system requirements in terms of building blocks representing a set of central functions which recur across many systems; to identify the tools and techniques which facilitate the construction of $C^3$ systems from building blocks; and to plan for a demonstration and subsequent implementation of the approach by the Air Force.

## CURRENT RESULTS

The need for reusable and adaptable $C^3$ data processing system building blocks has been reaffirmed. Some technical and administrative problems with the reuse of software today and the use of building blocks in the future have been identified. A technology assessment has indicated that a building block approach to $C^3$ system development and maintenance is feasible, provided a concerted effort is made to conduct tasks in the plan outlined in the report.

The initial concept of $C^3$ system software building blocks has been expanded from that of reusable off-the-shelf software modules to include other reusable products of the system development process. Requirements building blocks and design building blocks have been defined. They are identified as important elements in a collection of capabilities designed to reduce system development time and cost through increased utilization of existing products of the software development process.

A requirements building block is a statement of requirements for a data processing function that can be used as a starting point for the specification of the function in different systems. It defines functional concepts, terminology, and typical and alternative capabilities included in the function. The content and

form of requirements building blocks are being determined in the process of developing a building block for the $C^3$ system man/machine interface support function.

Three kinds of design building blocks have been identified: design structure, macro-design modules, and detailed design modules. The key to the development and use of reusable design building blocks is a methodology for partitioning systems into components and an associated design structure. A data flow architecture is proposed as the means for achieving flexibility and providing a framework for defining lower-level design building blocks.

Since some kinds of software building blocks are available today, it is suggested that experience with their application in $C^3$ systems be obtained and fed back into the requirements and design building block development efforts. A laboratory demonstration is identified as a means to obtain some of that experience and to evolve an approach to constructing systems out of building blocks.

## SECTION II

## BUILDING BLOCKS FOR $C^3$ SYSTEMS

### INTRODUCTION

In this section, the relevance of building blocks to $C^3$ system acquisition and maintenance is discussed. First, the characteristics of $C^3$ systems which establish the requirements for their computer systems are discussed. Important reasons are presented for investigating the feasibility of using building blocks in $C^3$ systems. Impediments to widespread use of building blocks for $C^3$ systems are also identified. Finally a revised concept of building blocks for $C^3$ systems is proposed.

### CHARACTERISTICS OF $C^3$ SYSTEMS

#### What is a $C^3$ System?

The discussion of an approach for building $C^3$ systems clearly requires an understanding of what a $C^3$ system is. The $C^3$ systems procured by the Air Force consist of heterogeneous collections of aircraft, sensors, communication equipment, data processing equipment, software, maintenance capabilities, shelters, furniture, and whatever else is required to provide a set of capabilities to fit into an existing or otherwise procured mission environment. They support such military operational functions as surveillance data acquisition, tracking, identification, weapons control, navigation, mission planning, operations monitoring, weaponeering, communications, network control, etc. The information systems that perform the military functions consist of people and procedures as well as the data processing hardware and software elements of the larger systems. The characteristics of the procured systems and of the information systems that utilize them affect the characteristics of the data processing systems for which one might define building block capabilities. Although the missions supported by $C^3$ systems may vary greatly from system to system, and although the details of the support for a given function may vary significantly from one system to another, nevertheless there are some common characteristics of the $C^3$ systems that result in some important common characteristics among the data processing systems which support them. Some important characteristics of these systems are as follows:

- A $C^3$ system frequently supports several military functions.

- There are frequently several instances of a system or of information processing system elements of a system. An instance may be in a fixed location or may be mobile.

- There are multiple external elements with which the system interfaces and many different message types exchanged over different kinds of communication media.

- There are requirements for real-time interface with the external world and for real-time internal data processing.

- There are requirements for continuous operations.

- There are frequently requirements for different modes of operation; for example, live and simulated modes, normal and expanded modes, or normal and degraded modes.

- They are semi-automated at best, and consequently human beings play a large role in the performance of the military functions. There is therefore a significant requirement for on-line interaction between the human and data processing elements of the system.

Some resultant requirements and characteristics of the $C^3$ operational software systems (data processing systems) are the following:

- They are large and complex because of the aggregation of several individually large and complex functions into a single system. The functions supported by a single system are generally related and as a consequence require significant internal interfacing. This interfacing is frequently effected through a common data base.

- They have requirements for adaptability of the software to:

  - different hardware configurations (resulting from differences between various instances of a system or from requirements for degraded operations or hardware backup),

  - different geographical locations,

  - different data environments, and

- different modes of operation.

- They have requirements for high software reliability because of the requirement to support the continuous operations of the information system.

- They have requirements for correctness because of the nature of the missions supported (for example, the classification or the direct life support characteristics of the mission).

- There are many different external messages accepted and transmitted. There are also many different application elements in the data base.

- There are requirements for real-time data processing.

- There are requirements for the quasi real-time support of multiple different user functions. As a consequence, there are many different user inputs to be processed and displays to be generated.

## THE IMPORTANCE OF THE BUILDING BLOCK APPROACH

To many people, a building block approach to system development means reutilization of software modules in more than one system. Many cogent arguments have been given for the use of this approach in DoD defense systems. For instance,

> "Having reusable software available can significantly reduce system development time. The more software that can be obtained off-the-shelf, the less new software that must be created. The risks involved are thus reduced, since off-the-shelf software should already have been well tested and debugged. Reduced time and risk enhance the probability that a project will be completed on time and within budget... Such benefits constitute a major resource savings, ultimately reducing the required DoD software investment." [COOP75].

Recently, a Congressional Committee, deliberating the FY78 defense budget, reviewed 18 different automated message handling systems. The Committee recommended drastic reductions in Research and Development funds for these systems because it felt that there

may be unnecessary duplication in their development. According to the Committee, fewer systems with higher degrees of commonality may be possible and desirable. While software is only one form of commonality which might be achieved, the magnitude of the savings which might result from reusing software was suggested by another source:

> "...transferring software from existing systems to new systems by one percent more would save $20M per year." [NYMA77].

The same Committee had another important concern about defense systems. It noted that it is not entirely clear that these expensive systems are designed to meet the needs of operational commanders. In many instances, the Committee stated, the operational commanders are being given information systems which they had little hand in developing and whose capabilities are frequently not fully understood and utilized. While the committee did not say so, a software building block approach may even improve this situation. A study of software problems by DoD for the House Armed Services Committee in 1975 recommended that we

> "...stop treating software as a fixed product which is developed, maintained and discarded; and begin to develop evolutionary systems which adapt and become more sophisticated over long periods of time." [CARL76].

If the use of software building blocks can significantly reduce the time for developing and modifying system software, then it would be possible to use an iterative, evolutionary approach to system acquisition. Users would receive operational capabilities through a succession of operational system deliveries such that:

- Users acquire an operational capability sooner;
- Users receive feedback on the adequacy of the requirements they have defined;
- Both <u>requirements</u> and <u>systems</u> can evolve to keep responsive to user needs in a cost effective way.

Pressures to reduce the duplication in system costs, to lower the cost of software, and to provide systems which are more closely matched to the operational needs of commanders, have all been cited as reasons for employing a building block approach to $C^3$ system

acquisition and maintenance. Pressure also comes from the direction of hardware technology. Smaller, cheaper microprocessors are causing a revolution in the architecture of computer systems. Networks of cooperating processors are threatening to replace monolithic systems with centralized processing. The promise of "logic on a chip" also portends a shift in the distribution of functional capabilities from software to hardware. These trends in hardware technology are a pacing factor in forcing the consideration of system structures composed of physically separate, possibly standardized, components. The economic advantages of such configurations will become irresistible. However, preparation is needed to design and implement systems which can exploit the distributed system architecture. A building block approach is one way to prepare for the use of standard functional components, which might initially be implemented in software, and eventually in some other form.

## PROBLEMS WITH THE USE OF SOFTWARE BUILDING BLOCKS

With all of the compelling reasons, cited earlier, for sharing software among $C^3$ systems, it is natural to expect that $C^3$ systems are frequently constructed this way. At present, this is rarely the case. There have been both administrative and technical deterrents to the use of building blocks. These are discussed below.

### Technical Problems

According to a DoD study of software problems:

> "There are two primary impediments to the widespread sharing of standard software modules. First, there is no satisfactory way to specify how a module will respond to all possible inputs. Second, such general purpose subroutines may not be structured in the most efficient way to fit in the context of the larger application system..." [CARL76].

Rigorous specification of module functions and adaptable software are certainly key requirements for sharing software. The current concern for improving the reliability of software has led to more formal specification languages for describing the functions of software modules. Specification languages can also permit selection of software modules for reuse in other systems. Of course, the accurate specification of what a module <u>should</u> do is of no value unless the module actually behaves that way, and we assert with

great certainty that it will continue to behave according to its specifications. Proving the correctness of software is a goal that current technology is approaching; the goal may be easier to attain for small, self-contained software building blocks.

The second impediment to widespread sharing of software, cited in the previous quotation, is more complex and difficult to overcome. For a module to be useful in more than one system, it may require adaptation in one or more of the following ways:

- The functions that it performs must be the right combination for that system, i.e., those functions that are required, without additional functions which are either unnecessary or already performed by other components of the system.

- The processing of those functions must be sufficiently fast and sufficiently accurate, with a sufficiently large capacity to meet the system load.

- The module's resources, such as computer memory, other hardware devices, and data bases, must be adapted to what the new system can supply.

- The software module must adapt to the architecture of the system, i.e., its internal interfaces, and the machine architecture. In other words, the software must be 'transportable'.

Adaptability in all of the above ways is difficult to achieve, and it can be an expensive process. Greatest success comes when the adaptability was planned at the time the module was designed, e.g., general purpose data management systems. Most general purpose software modules are far less efficient than special purpose software. A number of research activities in software technology may lead to the ability to describe a generic set of capabilities which, by semi-automated methods, is transformed into efficient software for a wide range of operational applications.

## Administrative Issues

Non-technical shortcomings have also made it difficult to reuse software from system to system. Administrative actions are necessary to facilitate planning for the use of common software in $C^3$ systems. Without such actions, it is less likely that a higher

degree of commonality will be achieved. Among the major administrative issues that must be resolved are the following:

- Major defense system acquisitions are each separately managed. Differences in funding and schedules alone can make it difficult to attempt a joint effort by several programs to develop common software. The initiative is best taken by some separate, central organization which could then make software modules available to all programs.

- To maximize the sharing of software components among systems may require some greater level of standardization in such areas as requirements, system interfaces, programming languages, and computers. The usual issues regarding standardization apply in this case too.

- Program managers must be willing to trade the desire for the most efficient solutions to their operational requirements against the lower cost and risk in using available but less efficient software components.

When software modules are available, the Air Force must be willing to require their use as a part of a new system. Such a commitment implies willingness to take responsibility for the performance of the government-furnished programs (GFP) and to maintain and upgrade them.

## ANOTHER CONCEPT OF $C^3$ BUILDING BLOCKS

In the context of $C^3$ systems, it may be beneficial to broaden the concept of building blocks from reusable off-the-shelf software modules to include other products of the system development process which may be adapted for reutilization among systems and between modifications of a system.

Two additional kinds of building blocks are proposed: requirements building blocks and design building blocks. Each of these is briefly described below and discussed in greater detail in subsequent sections of this report. The significance of this concept is explained below.

## Requirements Building Blocks

Operational requirements for a $C^3$ system describe the user's view of what functions a system will perform, how fast the functions must be performed, and other system characteristics such as reliability, availability, and survivability. Strictly speaking, requirements specifications do not define how the processing will be accomplished.

There is a level of commonality among $C^3$ system requirements. Requirements building blocks can be specifications for common functions. These specifications would be reused, with suitable modification, as parts of system specifications which require the functions they define. An example, elaborated on in the next section, is the operational requirement for a man/machine interface to an information processing system.

## Design Building Blocks

System requirements must be translated into a system design as a part of the development process. A system design is a structure of processing components and data bases, along with their interrelationships. Some portion of the total system requirements must be allocated to each component such that all requirements are satisfied. Designs are developed at several levels. A high level design simply partitions the system into a structure of functional components, while detailed design defines how each component will function. Modern software development practices begin with the high level, more abstract design, and gradually evolve details such as the selection of hardware and a system configuration, and the choice of processing algorithms, data structures, and data representations. Design building blocks can be reusable products of any level of the design process. These products include the organization chosen for a collection of processing components, as well as the design specifications of the individual components themselves.

## Significance of the Concept

The process of developing software for Air Force defense systems follows a sequence which is shown in Figure 1. Software is the end product of the development process which must be derived from the specification of requirements and the determination of a suitable design for a hardware/software configuration. Reutilization of software modules must be tied to both the requirements they fulfill and the framework provided by the system design. Hence the use of requirements building blocks and design

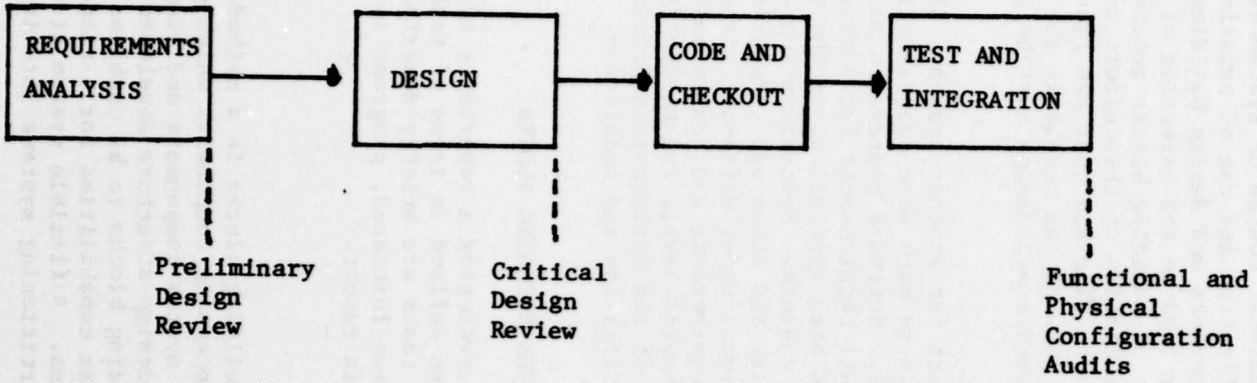



Figure 1. Phases of Software Development in the Air Force

building blocks follows the natural course of system development which can lead to the identification and use of existing software modules. Furthermore, requirements and design building blocks can be useful whether or not they lead to the selection of available software modules. Both types of building blocks represent significant effort. When duplication of that effort can be avoided from system to system, then both time and cost are reduced for system acquisition. Similar savings can result in the modification of systems, since the requirements and design must be altered to reflect system changes.

The concept is significant for another reason. Requirements and design building blocks may be much more adaptable to shared use than software building blocks. Software represents an optimized solution to a set of functional requirements for a given system. Furthermore, the software has been specialized to the hardware environment in which it will operate. Specialization and optimization reduce flexibility and often add complexity which makes it harder to utilize the software under different circumstances and harder to change it. Both requirements and design building blocks can be specified at a more general level, independent of details usually determined as a part of the optimization process. This should facilitate their reutilization and adaptation.

## A PLAN FOR DEVELOPING $C^3$ SYSTEM BUILDING BLOCKS

A preliminary plan for developing a repertoire of building blocks for $C^3$ systems has been defined in terms of tasks which should be undertaken. These tasks are briefly described below. For those tasks which have been initiated, progress is described in subsequent sections of this report.

### A Flexible Design Framework

The key to the use of building blocks is a methodology for partitioning systems into processing components which provides maximum independence between system components and simplifies their integration and testing. A design structure developed with such a methodology would allow building blocks to be combined in many ways to provide a variety of system capabilities for a range of different system hardware configurations. A flexible system structure and criteria for functionally partitioning systems into it must be developed.

Specification of $C^3$ Building Blocks

The objective of this task is to identify and produce specifications for information processing functions common to tactical $C^3$ systems. Requirements and corresponding design building block specifications will be produced. The design building blocks will be partitioned from the system design according to the criteria developed for the flexible system structure in the preceding task.

Design Trade-off Study

The objective of this task is to define tools and methods for utilizing design building blocks in the implementation of a system. Starting with a flexible system structure, developed in the initial task, detailed design trade-offs can be made to select a suitable combination of hardware and software. Analytical aids, such as simulation, can be adapted to the flexible system architecture to provide performance prediction. Other trade-off criteria, for which methods and aids are needed, include system reliability, cost, and survivability.

Support Tool Collection

The objective of this task is to develop an integrated collection of support tools to be used in the implementation of building block systems. Many of the tools may be simple adaptations of existing tools and the task becomes one of assembling and interfacing the tools with each other. Support tools encompass libraries for maintaining building blocks, design aids for selecting and combining and optimizing the performance of building block systems, tools developed in the Design Trade-off study, and tools which assist in the generation of software or hardware logic such as compilers for Very High Level languages, compilers which can tailor code to the parameters of a specific use of a building block, tools for timing the performance of the system, and tools for testing or validating both design and implementation of a system.

Prototype Demonstration

This task will demonstrate the application of the products of the other tasks to a set of tactical functions. Using the support system and prototype building blocks, an operational capability will be developed and its adaptation to a range of requirements demonstrated and measured for implementation time as well as system performance. If successful, this task proves the feasibility of the concept and opens the way for the procurement of other building blocks.

## SECTION III

## REQUIREMENTS BUILDING BLOCKS

A requirements building block is a statement of requirements for a data processing function that can be used as a basis or starting point for the specification of the requirements for that function in more than one $C^3$ system. The concept of requirements building blocks grew out of some specific needs in the ISS project and also out of observations about problems in the requirements determination, specification, and review process for $C^3$ data processing capabilities. The ISS needs were to identify data processing functions common to several $C^3$ systems and to express their requirements in such a way that they could be used to identify and ultimately acquire software building blocks. Needs seen in the general requirements specification area were for the development of a common understanding of what capabilities a given function may include; establishment of a common terminology for a given functional area; and documentation of the functional description so that it could be used as a starting point in the requirements determination process for a specific system.

System specifications (Type A) and development specifications (Type B) for specific $C^3$ systems have long been used as starting points or building blocks in the development of requirements for other related systems. For example, the specifications for each generation of the Air Defense systems have drawn heavily on the specifications for preceding systems. The availability of detailed, proven specifications from earlier systems is widely believed to be an important factor in the successful acquisition of a new system. The use of such specifications as models for new specifications has several limitations, however. First, a specification for a specific system reflects only one set of choices with respect to the capability offered for a given function. Alternatives are not identified, nor are the omissions, deliberate or otherwise. Also, the functional capability requirements are usually documented in terms of the specifics of the application. The potential applicability of a set of requirements to other systems may be obscured by the specificity, or simply by the use of parochial terminology. The development and use of requirements building blocks is seen as an extension and formalization of the practice of using the requirements - general concepts, details and words - generated for one system in the development of others. It is hoped that requirements building blocks will be more useful models than

individual system specifications because they will consolidate information about a given function from many sources; they will reflect options, and they will be expressed in generic rather than system-specific terms.

The concept of what a requirements building block actually consists of and what it looks like in detail is still under development. Our current activities directed at developing a $C^3$ man/machine interface support building block and an example of something like a building block are described under "How to Build a Requirements Building Block." Our current thoughts and assumptions about building blocks in general are described below.

## WHAT IS A REQUIREMENTS BUILDING BLOCK?

As was stated above, a requirements building block is a statement of requirements for a data processing function that can be used as a basis for the requirements specification for that function in more than one $C^3$ system. The functions for which one would have building blocks would obviously be those common to several $C^3$ systems. They might be large and complex, with many possible subfunctions, as for example, a data base management function. They might be relatively small functions, as for example, a simple keyboard editing function. They would include both mission-specific functions, such as tracking or identification functions, and mission-independent functions such as message processing or display management functions. As the functions vary in size, complexity, and logical relationship to each other, so would the corresponding building blocks. It is assumed that the functions and subfunctions for which building blocks are defined will be determined by a methodology that results in logically coherent functions with well-defined external interfaces.

The requirements are understood to be data processing requirements, not software requirements. No assumption is made about the allocation of those requirements to hardware or software.

Requirements building blocks are documents, written according to some yet to be defined organization and content specification. There would be some descriptive and explanatory material and some data processing function specification material suitable for easy incorporation into requirements specifications. In the long run the specification portions might be written in a formal requirements or specification language.

Requirements building blocks are seen as being vehicles for the expression of functional requirements primarily. Although it might be possible to indicate the kinds of performance requirements appropriate to a function, it seems inappropriate to try to indicate a function's performance requirements quantitatively out of the context of a specific system. A requirements building block might also include a mechanism for describing the critical characteristics of the function and the contexts in which it might be used so that its applicability in a given situation could be readily determined. It would also be desirable to provide a mechanism for identifying high level characteristics of the requirements and the corresponding set of appropriate subfunctions.

The requirements contained in a requirements building block are for capabilities visible to and accessible to the external interface of the function. In the work on the man/machine interface support function described below, the capabilities are those visible to the on-line system user. Clearly, not every data processing function for which there might be a requirements building block has an external human or other system user. One of the most promising areas for the development of building blocks is in the intermediate level between the application specific external system interface and system architecture and hardware specific functions. So, users of a function must be seen as programs as well as human beings and other systems.

## HOW WILL REQUIREMENTS BUILDING BLOCKS BE USED?

A requirements building block will be used to help define the requirements for a given function in a specific system. It will also be used as the statement of requirements for the development of a matched set of design and code building blocks.

Requirements building blocks might be used in the following way. During the determination of the overall requirements for a data processing system and the identification of the functions to be performed, the set of available building blocks might be used to suggest possible (and presumably reasonable) functional partitions. Given a functional decomposition for a given system, individual building blocks could be used in the following ways by someone specifying detailed data processing system requirements. For every function for which a building block having reasonable correspondence exists, the building block will be reviewed. The purpose of the review is to identify (or refresh the memory of) the concepts, terminology, and capabilities normally associated with that

function. The overall structure of complex functions will be seen as well. The specification for the function in the specific system will use the building block concepts, terminology, and functional structure. A subset of the building block capabilities will probably be identified as those capabilities meeting the system's requirements. It is assumed that the building block will show functional dependencies and that the completeness of the selected subset could be checked. For a given capability, the building block may indicate several ways in which the capability can be provided; the desired ones would be selected. For some capabilities the statement of requirements contained in the building block will be copied without change. For other capabilities, they may need to be augmented or modified to meet the system's requirements. The specifier will develop the specifications for requirements not expressed in the building block, keeping them as consistent with the rest of the specification as possible.

In an automated ISS environment, the building blocks could be expressed in a formal machine checkable notation. The automated support to the specification process would provide for the creation of new specifications from the building blocks. It would include checking of the new specifications as well as general text editing and manipulation. It might also include such things as the identification of corresponding design building blocks. As user requirements evolve they will be incorporated into the system specification, using the building blocks as a baseline for their definition and expression.

Requirements building blocks can be used for other purposes in the system development cycle besides the specification of detailed data processing system requirements. Specifically they can be used in the review of the specifications by other than the preparers. The building block would provide a checklist for checking completeness and for reminding the reviewer of alternatives rejected. They can also be used as a framework for comparison of systems when that is of interest.

Requirements building blocks will also establish the set of capabilities for which design and code building blocks are built in an ISS facility.

## HOW TO BUILD A REQUIREMENTS BUILDING BLOCK

An obvious activity in an effort to develop $C^3$ system building blocks is a review of existing or specified $C^3$ systems to assess the commonality of system functions and their data processing support requirements. Although we recognized from the outset that a comprehensive review and analysis of $C^3$ system characteristics were beyond the scope of project resources available for such a task, we nevertheless believed it necessary to develop some insight into the degree of commonality across $C^3$ systems and into the difficulty of obtaining that kind of information. Our resources were more limited than expected and the job turned out to be even larger and more difficult than anticipated. As a result we are unable to make conclusive statements about the extent of the commonality of data processing support requirements across $C^3$ systems, but we do have some greater insight into the subject. We also believe that we have made some progress in the development of a requirements building block for the man/machine interface function in $C^3$ systems.

The approach taken was to look in a top-down manner for commonality of processing requirements as identified in System (Type A) and Software Development (Type B) specifications, and similar documents. Available documentation for 12 ESD systems was briefly reviewed. The systems are identified and categorized below.

Communications Systems

AFSATCOM I
JTIDS
SATIN IV
TRI-TAC TCCF
TACS/TADS

Surveillance and Control Systems

E-3A
COMBAT GRANDE
JSS
GEODSS
PAVE PAWS

Command and Management Systems

TACC AUTO

Intelligence Systems

TIPI-DC/SR

The review revealed that communications processing and on-line user/machine interface support functions were the most commonly occurring requirements across these systems. A more detailed look at those two functional areas was then initiated to obtain a better understanding of the degree of commonality in those areas. A subset of the 12 original systems was selected for the next level of assessment in each functional area.

The systems selected for further examination of the communications processing requirements included communications, surveillance and control, and command and management systems. They were the following:

AFSATCOM
JTIDS
SATIN IV
JSS
TACC AUTO

Working from requirements level documents (primarily Type A specifications) it was possible to categorize the communications software processing requirements into the five areas of message processing; network control; communications equipment interface/control; operator interface; and performance monitoring. Subfunctions in each area were identified to provide a mechanism for grossly representing and comparing requirements across the systems. The tabulation of requirements across the systems showed the presence of many of the subfunctions in several or all of the systems. However, although such results could be seen as an indication of potential commonality, the review tended to inspire skepticism by the reviewer about the presence of true commonality, around which building blocks could be developed. The determination of the requirements and their comparison across systems were, as expected, made difficult by the varying levels of detail in source documents and the differences in terminology from system to system. The systems were deliberately selected to include those in which communication was a primary function and those in which it was not. The communications processing functions in these systems support very different communications system architectures, processing and transmission equipments, protocols, and message types. These are some of the communications system characteristics which clearly can influence the nature of the detailed requirements for data

processing support. The effects of these characteristics may not be apparent at the level of documents reviewed and to the extent that they are, they seem to reflect more difference among the systems than commonality.

The review of the communications area was terminated after a short effort. The primary result of that effort was a heightened appreciation of the apparent differences among the communications areas of $C^3$ systems as a whole and the size and difficulty of the task of further assessing the potential for meeting their requirements with common building blocks.

The further examination of the on-line man/machine interface requirements within $C^3$ systems was pursued in two ways, one looking for gross indications of commonality among $C^3$ systems, and the other asssuming some commonality and aimed at the development of a common framework or model in terms of which the requirements of diverse systems for man/machine interface support could be expressed.

The review that was intended to give gross indications of commonality among systems for man/machine interface support requirements included COMBAT GRANDE, representing the air surveillance and control class of systems; TACC AUTO, representing the command and management class of systems; and TRI-TAC, representing the communications class of systems. The source material available spanned Type A, B, and C specifications and obviously varied in the amount of detail presented. The requirements information extracted included the following: the generic types of devices included in the user station (CRT graphic and/or tabular display, alphanumeric keyboard, various types of switches, cursor positioning devices, printers, audible alarms); the various types and numbers of information displays presented (graphic, tabular; fixed, ad hoc; requested, forced; substantive, forms; etc.); the numbers and types of screen areas defined for independent and concurrent displays (tabular, situation, attention, error feedback, menu, etc.); display manipulation capabilities (off-set, expansion, feature selection, etc.); and number and mode of user inputs. The primary result of this review was observation of enough commonality of requirements to support further interest in continuing the separate and concurrent effort described below.

The second effort in the man/machine interface requirements area has been in the development of a model for expression of those requirements. The objectives of the effort have been the following:

- To identify grossly the characteristics of an interface between the data processing system and the on-line user that can be used for the purpose of facilitating top-level differentiation among interfaces and permitting inference about the data processing support required for the interface.

- To identify the set of on-line user requirements and capabilities that are present in $C^3$ systems.

- To categorize and organize those requirements.

- To identify options available in the selection or specification of capabilities to meet higher level requirements.

- To provide common concepts and terminology for use in the description and specification of a man/machine interface and its data processing support.

- To provide a vehicle for assessing commonality among different interfaces.

- To provide a checklist for preparing or reviewing man/machine interface support specifications.

- To provide the statement of functional requirements against which man/machine interface design and code building blocks could be developed.

The on-line man/machine interface area was selected as the subject of the requirements building block effort because of the universality of man/machine interface support requirements across all $C^3$ system types and the resultant potential for broad application of the results. The choice has also been inspired by the importance of this functional area in user acceptance of and ability to use the data processing systems. It was further motivated by interest in movement towards some human/system interface standards. It was also felt that making requirements explicit might correct the problem of underspecifying and therefore underestimating the implications of man/machine interface requirements.

The initial work on the model consisted of a detailed review of the TACC AUTO Type A specifications to extract the man/machine interface requirements. The requirements for the interface support

effected through the tabular display terminal were analyzed and used to formulate a simple model for organizing and categorizing those requirements. The model included functional, capacity, and performance requirements. The functional requirements categories were the following: management of displayed data, management of display screen content, man/machine protocol, ease of operator use, formatting, and editing. The COMBAT GRANDE interface requirements were subsequently extracted from the Type B specifications and organized in terms of the model to test for general suitability of the model to the requirements of a very different $C^3$ system and to compare the specific requirements of the two systems. The COMBAT GRANDE specifications mapped into a subset of the TACC AUTO based model. The absence of any COMBAT GRANDE requirements to support the management of displayed data (e.g., to store a display or to transmit it) highlighted the difference between the two systems in the use of the systems and the display interface by the on-line users. The other functional categories generally accommodated the COMBAT GRANDE requirements reasonably well. A comparison of the requirements of the two systems reveals some obvious commonality - such as editing requirements - and also indicates a potential for identification of greater and higher level commonality given a common terminology and level of detail in the expression of the requirements.

The second phase of this effort has consisted of a significant broadening of the scope of the model, reorganization and expansion. The approach that has been taken in this second phase is to begin by developing a strawman idealized model covering known types of requirements from many different systems. The next step is to then refine it by mapping the man/machine interface characteristics and requirements of various $C^3$ systems into it and iteratively tuning it to better reflect those requirements.

The current version of the man/machine interface requirements characterization is contained in Appendix I. It is a preliminary general model that is being used in the review of existing $C^3$ man/machine interface requirements and has yet to be modified to reflect that review.

The first section of the model, Overall Characteristics of Position, identifies the overall characteristics of the user's role and environment that should be recognized and may affect the nature of the data processing support requirements. The second section of the model, Workstation Characteristics - Physical Environment, identifies physical characteristics of the user station and its

environment that need to be considered in the specification of the interface.

The third section of the model, Software/Processing Requirements Directly Related to User Interaction, addresses the functional requirements of the man/machine interface. It is subdivided into components that identify capabilities for significantly different aspects of the interface. The first subsection identifies the user's requirements for management of the medium of the interface. For example, it identifies capabilities for controlling display content, appearance, space utilization and allocation, etc. The second subsection identifies generic capabilities needed to support the user in the performance of his task. Data editing, data entry, and data transmission are among the kinds of capabilities included. The third subsection identifies user requirements for error handling and the fourth subsection suggests types of capabilities that may be provided to facilitate the user's interaction with the system. The last section, Software/Processing Requirements Indirectly Related to User Interaction, identifies data processing system requirements that are not considered to be a part of the man/machine interface requirements but are related to them.

This man/machine interface characterization is far from being a man/machine interface requirements building block for $C^3$ systems. It is still a general model, not yet specifically tuned to $C^3$ systems. It needs significantly more work than has been put into it so far to clarify man/machine interface concepts and terminology; to elaborate in general and even out the level of detail of the capability specifications; and to select and implement an appropriate orientation and general form of the requirements statements or specification. The characterization has been included here because it is the best approximation of a requirements building block that we have developed thus far and provides some illustration of the concept. It is also felt that even in this preliminary state it may be a useful tool in the development or review of man/machine interface specifications for $C^3$ systems currently being acquired. It is suggested that it could be used to review the completeness of specifications and to indicate alternate ways of providing capabilities.

## SUMMARY AND CONCLUSION

Our work in the requirements area this year has convinced us that the development of requirements building blocks for $C^3$ systems

will require significant amounts of time and effort. Further ISS work should make it possible to procure requirements building blocks. The completion of one building block can serve as a model for others. Other tasks should also include the development and documentation of the general definition of a requirements building block and an approach or methodology for generating specific instances of one. In addition, the building block should be used as the statement of requirements for the procurement (presumably the design and implementation) of at least some portion of a prototype man/machine interface capability to assess its usefulness and to provide input to its refinement.

## SECTION IV

## DESIGN BUILDING BLOCKS

### INTRODUCTION

This section relates the design process to the phases of $C^3$ system acquisition and suggests several types of design building blocks. The key to the use of design building blocks is a methodology for selecting system components and organizing them into a system structure. A data flow architecture is proposed as the means for achieving flexibility and enhancing the reusability of functional design building blocks within and among $C^3$ systems.

### THE DESIGN PROCESS FOR $C^3$ SYSTEMS

In a sense, the design process can be defined in terms of Air Force regulations governing the acquisition of computer resources. These regulations dictate products of the design process which must be delivered for review and approval, as well as the sequence in which these reviews occur. Air Force Regulation 800-14, Volume II [AIRF75], describes two phases of computer program development during which design is accomplished. The Analysis phase normally begins with the release of the system specifications and ends with the successful accomplishment of a Preliminary Design Review (PDR). The Design phase which follows accomplishes the development of a preliminary Computer Program Product Specification and its review during the Critical Design Review (CDR). Thus, there are two major stages in design. We will refer to the first as "macro-design" or "preliminary design" and the second stage as "detailed design."

Overall, the design process is one of transforming system requirements into a structure of system components which together accomplish the required functions. The design process is a succession of similar activities, which partition the system into components at finer and finer levels, allocate requirements to components, and define interfaces. As summarized by Glore [GLOR77], during preliminary design the system specifications are divided into components called Functional Areas, interfaces are defined among components, and the system's functional requirements, technical performance, external interfaces, design and test requirements are allocated among the Functional Areas. Functional Areas can be divided into segments, if necessary, and then into Configuration

Items (CI) or, in the case of software, Computer Program Configuration Items (CPCI). Finally CI's can be divided into functions. As the design process proceeds, algorithms are selected and computer resources are allocated to functional components.

## THE IMPORTANCE OF DESIGN

Design decisions affect the technical quality of a system and the ease with which a program can be managed. The partitioning of the system at the macro-design level affects the division of labor for developing a system. Separate organizations or groups within organizations may be responsible for developing individual subsystems or CI's. The level of communication among these organizations will be affected by the number and complexity of interactions among system components. The number and composition of Configuration Items affect the number of design reviews, the difficulty and extent of the system integration and testing activities, and the level of the government's technical monitoring of full-scale development.

Design decisions affect the quality of software in several ways. A study of software errors [BOEH75] showed that 65% of all errors found were design errors, that is, errors that required changes in design. The ease of integration and system level testing, and the rate at which causes of errors can be identified are affected by the number of dependencies among system components. In a similar way, the effort required to modify a system will depend on the distribution of functions and interfaces among its components.

## DESIGN BUILDING BLOCKS

Any component of a system which is specified as a part of a system design can be a building block. At the macro-design level, the building block may be specified as a black box, whose external attributes only are described. External attributes include external interfaces such as input and output formats and data requirements, functions performed and output generated for each legal set of inputs, and errors detected. At the detailed design level, the algorithms for accomplishing the functions of a building block are defined as are the data structures used for the processing. These detailed design specifications constitute a second type of building block.

There is one additional product of the design process which is essential to the reusability of designs: the structure of the system design. The design structure is derived from an explicit or implicit set of rules for partitioning a system into components. At the abstract or conceptual level, the structure is the counterpart of a system architecture, although it may differ from the actual system architecture which is implemented in terms of configurations of computers and other hardware, and the connections among them.

In summary, three kinds of design building blocks are possible: a design structure, macro-design modules (components), and detailed design modules.

## A FLEXIBLE SYSTEM DESIGN STRUCTURE

The importance of a flexible system design structure has already been noted. It is the key to partitioning a system into components which can be used as parts of other systems. It must also allow the alteration of a system to easily accommodate required changes in the system's behavior over its life cycle.

### Attributes of a $C^3$ Information System Structure

A system structure, like a network, is composed of components and connections among them. In an information processing system, the components consist of processes and data. A processing component also referred to as a module, accepts data as it is input, performs some set of functions, and produces data as output. The connections among components may be viewed as relationships of two types: control, and data. Control relationships determine what processes can cause other processes to execute their functions. Data relationships determine how the outputs of processes become the inputs to other processes. Connections among components within a system are internal interfaces. A system also has external interfaces to its operational environment. These interfaces can be stimuli which enter the system such as signals from equipment and sensors, messages from other systems or actions of operators. External outputs, similarly, can be messages, reports, displays, or signals to control weapons and sensors.

The design structure of a system can also determine the sequence in which processes are executed. Execution sequence can be a function of control flow and of data flow through the system. Sequential (or one step at a time) execution and concurrent (or parallel) execution are possible.

Many of the functions of a $C^3$ system are real-time functions, each activated by some external stimulus to which the system must rapidly respond. The total system requirements consist of many such stimulus-response patterns, often occurring asynchronously and concurrently but not independently. Thus the system can be viewed as a set of cooperating processes sharing data.

A Proposal for a Flexible System Design Structure

The following is a proposal for a design structure which might provide flexibility and serve as the framework for defining functional building blocks at the macro-design level. This proposal is in line with a variety of ideas appearing in many contexts for many purposes in current computer science literature. Some of these related ideas are described later in this report.

Major characteristics of the design structure, which will be referred to as a data flow architecture (DFA), are the following:

- The system is viewed as a collection of processes which can operate concurrently. Each process is itself a single, sequential process.

- The connections between processes are primarily data connections, i.e., the outputs of processes are connected to the inputs of other processes. The data are explicit, self-contained messages which are defined so that they can be tested for reasonableness.

- The sequence of execution of processes is determined by data flow through the system, i.e., any process may operate when it has inputs and cease execution when it has generated the corresponding outputs.

- Each process has its own resources. Messages are transmitted among processes but data resources are not shared. In concept, each process also has its own processor.

Such a structure is clearly abstract, in the sense that it ignores many of the realities of an implementation such as sequence control, synchronization of processes, and resource allocation. Application of the DFA also requires the development of a companion set of partitioning rules which provide guidance in the selection of the functions for each processing component.

At present, the DFA appears to have attractive advantages. Its controlled application to $C^3$ systems is needed to refine the concept and to develop the rules for its use.

Advantages of a Data Flow Architecture

The Data Flow Architecture is representative of one stage in the transformation of a set of system requirements into an implementation of those requirements. It is a stage worth capturing and documenting because it is the point at which the essential dependencies among components of the system design are shown in their simplest terms. It is free of dependencies and complexities introduced by implementation constraints. It is possible to move forward from the Data Flow Architecture in many directions toward an implementation. For example, processes can be centralized or distributed over more than one processor; processes can be scheduled or allowed to run concurrently and asynchronously. The use of concurrently executing processes in the DFA makes it easy to map the system design into either sequential or concurrent processing components, whereas it is more difficult to unravel a system design which assumes strict sequential order into one which is implemented with concurrent processing.

The restriction of connections in the DFA to explicit transfers of information among modules has several advantages:

- This corresponds closely to the stimulus-response nature of $C^3$ system requirements. System behavior, as represented by the design, can be traced back to and verified against system requirements which are expressed as expected outputs for sets of system inputs.

- All interactions between modules are observable. There can be no implicit side effects. Analytical aids can be used to test the data connections to see if they are well-formed, i.e., consistent and complete.

- Modules defined for a system using the data flow architecture should be easily reused. Their specifications can be expressed solely in terms of input, process, output. They are free of any assumptions about the behavior and sequence of execution of other modules so long as inputs to them are generated by the system. They also perform simple sequential processes, so they do not rely on complex control sequences.

System Implementation of a DFA Design

As noted above, a macro-design of a system using a DFA is one stage in the total development process. The utility of a DFA must be judged by how well it prepares for and simplifies the stages of development which succeed it.

If the architecture of the system implementation were identical to a DFA, then the implementation of a DFA design would be straightforward. There may soon be data flow-oriented distributed processing systems, composed of a network of processors, each dedicated to a single process with adequate resources for that process, and a very fast data bus over which processes send messages, containing requests or information, to other processes. Many elements of such a system are already feasible.

There are ways to approximate the DFA design in a system. Explicit data connections have been achieved between processes on a single processor through strict use of a message queue for interprocess communication. The queue can also be used to schedule processes which have messages, i.e., as determined by data flow. If data connections are more efficiently provided through sharing of data among processes, then rigid and explicit control over access to common data bases can, at least, minimize the possibility of unintended accesses and limit the extent of data dependencies. Such control over common data base access can be achieved by management policies, automated aids and language features. All of these methods must force explicit declaration of common data needs for each module, see that these are minimal, and test for conformance to the declarations. Control over data base access can be done during system operation, at system load time, or even by a compiler.

The DFA design does not deal with scheduling and sequence control of functions or with resource allocation. Much of the DFA system design can be preserved if these functions are introduced into an implementation as separately as possible from other system functions. This fosters a closer correspondence between the processing modules of the DFA design and the software modules which implement them. As long as the software isolates control dependencies, the modules are capable of greater utilization in other combinations and other configurations.

In the studies summarized below, mechanisms for closely approximating features of the DFA are identified.

## STUDIES RELATED TO DATA FLOW ARCHITECTURE

The idea of a data flow oriented system is neither original to this project nor of recent vintage. The association of a Data Flow Architecture with the objectives of this project was due to our familiarity with a suggestion made several years ago by a colleague. He proposed a new software design technique which he termed "egalitarian programming" [SULL73]. He felt it would increase the flexibility of programs and the clarity of expression of procedures within programs. He demonstrated by example that unnecessary dependencies were created among procedures by a hierarchical decomposition of a system into levels of control which determined which functions might call on which other functions. Many of these unnecessary dependencies could be eliminated by placing all procedures at the same level of control and allowing them to interact through the transfer of data alone.

Subsequent to selection of the DFA for the ISS project, a survey of recent computer science articles was conducted to see what kind of support or evidence might be found to confirm the utility of the DFA. A number of recent articles was found which presented similar architectures to serve a range of purposes such as improved system design, compiled code optimization, distributed processing system architecture, and a new computer architecture. A representative sample of these articles is briefly presented below. With increased interest in distributed and concurrent processing, more work on data flow analysis will undoubtedly appear.

### System Partitioning Study

The Systems Technology Project Office of the Army Ballistic Missile Defense Systems Command has initiated a System Partitioning Study, reported in [SMIT76]. The orientation of the study is toward improving the technical and contractual management of large system acquisitions by developing better rules for selecting major system components. These rules would be applied prior to detailed system design. The outcome would affect the boundaries between multiple organizations involved in a development effort. By inspection of several similar systems, and interviews with developers, a preliminary set of partitioning rules was deduced. These rules were used to repartition a portion of the Site Defense program. The study concluded that complexity, cost and company interactions might have been reduced by better partitioning. Several rules are concerned with partitioning a system into elements with interfaces "which feature messages to be acted upon and which are subject to reasonableness tests for error control..." and producing elements

"that exhibit the performance of a complete function based upon a clear stimulus and upon well-bounded output requirements." The application of these and other rules was termed "Data Flow Partitioning". The study also described the use of $N^2$ charts as a technique to document the data flow among processing functions and to assist in the decomposition process.

Modular Software Construction

The study which comes closest to the Data Flow Architecture concept is one by K. Jackson [JACK77]. This paper recommends using parallel processing as the basic criterion for decomposing a system into modules. The approach provides flexibility to combine modules in many ways in the construction of real-time systems. Process synchronization can be handled in a modular way by concentrating it in the access control to intercommunicating data areas. Precompiled modules can be formed into a network by naming the data areas required by each process. A system, called MASCOT, implements these concepts. It provides facilities to form systems from collections of highly independent modules with restricted forms of communication and connection. MASCOT also contains synchronization primitives to control data flow through a system. Its use is reported to have resulted in "remarkable" programmer productivity and ease of integration. A language, MORAL, was reported to be under development to use with MASCOT.

A Highly Reliable Distributed Computing System

A process structure similar to the Data Flow Architecture was proposed several years ago as a design for a hardware configuration of a highly reliable, distributed processing system [GORD73]. Major features of the design are: a decentralized processing environment in which there is no central control; separate and non-overlapping memory and input/output resources for processors; and connections among processes primarily through the sending of messages, rather than direct control. A model was developed which contains elements for controlling the system, connecting user functions, and handling files. Subsequently, a Distributed Computing System (DCS) was developed at the University of California, Irvine which implements the basic concepts of the model. [ROWE75] The DCS consists of processors connected to a unidirectional digital communication ring. Processes are assigned to processors and then interact with other processes by addressing messages to them and placing them on the ring. Functions to schedule processes within a processor and to perform message transmission are contained in a nucleus within each processor. While processes do directly address other processes by

name, they do not know the processors on which they reside. Allocation of processes to processors is a function handled by polling all processors on the ring. Resource allocation, input/output functions and file management are also distributed processes which are accessed by messages. Each resource is treated as a process which need not reside on the processor whose resources it allocates. Failsoft operation of the system is achieved by distribution of control, isolation of processes, controlled access, and redundancy. This prevents the failure of one component from causing total system failure.

Operational Software Concept

Several years ago, the Air Force Avionics Laboratory sponsored a study to develop a design methodology and framework for efficient reuse of operational flight software throughout the life cycle of an avionics system and to permit the efficient transfer of software between different missions and different computer architectures [ENGE76]. During the study, a software structure was defined, tools and documentation aids were selected to support its use, and its effect was evaluated by employing the design framework in the construction of demonstration software.

The software structure was based on partitioning rules to enhance reusability which included "separating out machine dependencies, mission dependencies, utility functions, data structures, and individual function elements." Interfaces between these clusters of functions are concentrated in specific system components. For example, the executive acts as an interface between application programs and the computer as well as real-time dependencies. An Intertask Communication component controls all synchronous operation, while a Data Access Control component handles asynchronous access to data. The study describes the use of Directed Flow Graphs, which provide a graphic notation for explicitly representing both data and control flow, and an associated algebra for mathematically verifying the operation represented by the graphs.

Through a prototype demonstration, the study showed that common software modules can be built for different missions and different computers. While the price of commonality was inefficiency, it was felt that hand tuning could overcome that problem. Furthermore, the unoptimized version is the best baseline from which to make system modifications or reapplications because it is in its most machine-independent, mission-independent form.

## Data Flow Computer

Recently, a data flow model was described which can serve as a conceptual tool to explore a new kind of computer architecture [RUMB77]. The model appears to be equally useful for system design, and is similar to the Data Flow Architecture proposed in this report. The model consists of a network of functional operators connected in pairs by one-way asynchronous data links. Modules (operators) interact by sending messages. The sequence of execution of all operators is independent and concurrent. Any operator is enabled when values are present on all of its input links. Side effects between modules are avoided by the same explicit links for both flow of control and of data.

## Modular Concurrent Programming

A recent paper described the results of research to develop an effective method for constructing large, reliable concurrent programs from small modules [BRIN77]. The programming language Concurrent Pascal was designed to support the method. Processes perform operations on data that are inaccessible to other processes and communicate via a special module called a monitor which controls access to shared data. The compiler checks data access as well. The effect is to isolate programming errors within modules and to facilitate systematic testing. Experiments in using the method and tools for three model operating systems showed greater efficiency in implementation of the systems and higher reliability of the software. A surprise benefit was noted: 14 modules of one system could be used without change in another system.

## Data Flow Analysis

There are a number of algorithms which have been developed for analyzing the flow of data in computer programs, e.g., [ALLE76]. These algorithms, based on graph theory, are being used for optimizing the execution speed of code produced by compilers. It has been suggested that they might also be used to find anomalies in data processing which are potential errors in program logic [FOSD76]. These techniques translate control sequences (a succession of operators accessing and changing data) into data flow sequence. These or similar techniques might be used to analyze system designs in data flow form as well.

## CONCLUSIONS

Design building blocks appear to be a useful extension of the building block concept. The concept of a Data Flow Architecture is worth investigating and refining because of its general utility in the design of reliable, modular concurrent processing systems. It is also an important key to partitioning systems into reusable components.

## SECTION V

## SOFTWARE BUILDING BLOCKS

A software building block is a capability for providing compilable or executable code for incorporation into $C^3$ systems. A software building block may take the form of an automatic program generator. In the short term, a $C^3$ system software building block is more likely to take the form of source code that is capable of being adapted or modified for use in various systems.

A technology review early in the ISS project identified some commercially available systems and some research and development systems that generate software. There are a number of business application systems that generate code, usually COBOL, given a specification more abbreviated and frequently much less detailed than a COBOL program. These systems include ADPAC, METACOBOL, SCORE, CL*IV and WORKTEM, summaries of which can be found in [AUER] and [CANN75]. It is difficult to learn the details of implementation of these systems but, in general, they appear to use existing COBOL code designed to fit into some basic standard structure, tailored to meet specific needs, and possibly augmented with user specific code. The Business Definition System, a research effort at IBM Thomas J. Watson Research Center, is aimed at generating business programs based on the answers to a multiple choice questionnaire that elicits user requirements. [HAMM74, HAMM75] A baseline application program for an application area is contained within the system as are the alterations to that program which correspond to the possible answers to the questionnaire. The system generates a customized program based on the model and the questionnaire answers supplied by the system user. Protosystem 1 sought to automate the analysis, design, and coding phases of software development for a class of data processing systems which perform simple operations on very large keyed files in batch mode. [RUTH76] The system takes as input a very high level non-procedural language, HIBOL, and after several phases of processing, generates optimized PL/1 code. It is noted that this work is all in the business application area and directed to the generation of code for well understood problems to be implemented in batch data processing systems. There is much interest in the development of automatic programming capabilities but it seems likely that code generating building blocks will not be available for use in $C^3$ systems in the near term. We assume then that ISS software building blocks will be in the form of code.

## SOFTWARE BUILDING BLOCK CHARACTERISTICS

Software building blocks do exist today. Examples are subroutine libraries or the individual subroutines within the libraries, operating systems, data management systems, report generators, etc. Software building blocks vary from the very small building block, such as a subroutine in a subroutine library, to very large, such as a modern operating system or data management system. A software building block may or may not be adaptable. A very small building block probably won't be adaptable because it is more work to adapt it than to write your own. The larger building blocks are frequently adaptable through such techniques as run time parameterization or conditional compilation. The small building blocks may be reasonably transportable. The large system building blocks, however, are very frequently tied to the hardware architecture and some well-defined range of hardware configurations on which they will run so that in fact the building block is a combination of hardware and software. The very small building blocks are plugged into a detailed program design and are used to reduce the amount of coding and unit testing necessary in the program development. The larger building blocks that provide a higher level of capability are generally selected early in a system development. They contain much of the detailed design for the capability they provide and affect the design of the system built around them.

Software building blocks not only exist but they are being used today in $C^3$ systems. Sizable pieces of both application-dependent and application-independent software from the 407L CRC operational and Recording Program were used in the TACS/TADS Interface Module Processor Program and those in turn have been used in the SALTY NET operational program.

The most commonly used building block in $C^3$ systems is probably the operating system. For example, in both the TACC AUTO communications processor and data source terminal elements, an existing operating system was used as the basis for the operational program. The PAVE PAWS central data processing system software is built around a modified version of a commercial operating system. The Standard Software Base (SSB) is a set of software building blocks that has been developed by RADC/IR for use in the development of systems to support intelligence analysts. [MIX77] The SSB is built on a commercially available operating system and provides building blocks that support display terminal and various communication system interfaces.

Software building blocks have also been used in $C^3$ testbed and prototype systems. The TACC Current Operations and Current Plans support software developed in the MITRE/ESD Tactical Data Systems Development Testbed were built on operating system, data management system, and display generation and management building blocks. A commercial data base management system was used in a prototype version of the ATEC system. The Navy/ARPA Advanced Command Control Architectural Testbed includes several kinds of software building blocks to support its development of $C^3$ capabilities. [NELC76] The MIT candidate message processing system in the Military Message Experiment was based on a library of existing software building blocks.

As was indicated above in Section II, there are problems with the use of software building blocks which have limited their success and have inhibited their widespread application. The problems listed there are the differences in functional requirements from system to system and the different functional environments into which a building block must fit; different requirements for speed, accuracy, and capacity; different hardware configurations and architectures; and different interfaces among software functions and between the functions and the data to be processed.

Although these problems are seen as being associated with the reuse of software, they are mostly not implementation problems. The fundamental problem with the reuse of software in the face of these differences is insufficient modifiability or adaptability. The problems associated with adaptability to different functional requirements, different functional partitioning, and different functional interfacing are basically design issues and should be addressed as design issues. Some of the implementation problems in the reuse of software are being addressed by the movement toward high order language standardization and toward establishment of standard computer family architectures. Pending further progress in the development of an ISS design approach, it appears that the most important activity that can be pursued with respect to software building blocks per se under the ISS program is the examination of the use of existing software building blocks in $C^3$ systems. This investigation should include both the review of experience with the use of software building blocks in those systems where they have been used, and the laboratory application and demonstration of software building blocks. The objective of such an investigation is to develop insight into the use of software building blocks which can be fed back into the development of a building block methodology and design structure. It would allow the determination of what shortcomings the current software building blocks have relative to

real $C^3$ requirements. The shortcomings in both basic capabilities and adaptability would be identified. In particular, performance problems would be examined and ways of addressing them would be sought.

## RELATIONAL DATA BASE DEMONSTRATION

As a part of the FY77 ISS program an effort to demonstrate the application of a software building block to a $C^3$ system data base management problem was initiated. The data base management function was chosen because it is recognized as an important data processing function in many $C^3$ systems and because it is a function for which building blocks are available. As was indicated above, data base management packages have been incorporated into some prototype $C^3$ systems. The system selected for application is the INGRES system, which operates under the UNIX operating system on the PDP 11/45 computer. Although the effort is still in its early stages and has therefore not produced any results that can be reported, it is useful to discuss the rationale for the selection of INGRES and the objectives of the effort.

INGRES is a relational data base management system, as distinguished from a hierarchical (e.g,. IMS/VS) or a network (e.g., IDS) system. A relational view of data

> "provides a means of describing data with its natural structure only - that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other" [CODD70].

The primary advantages of relational systems have been summarized in [CHAM76] as simplicity, which allows a user to formulate requests (or operations) simply in terms of information content; data independence, the immunity of applications to change in storage structure and access strategy; symmetry, which makes the ease of asking a question at the user interface independent of the structure of the data base; and its strong theoretical foundation, which rests on the mathematical theory of relations and on the first-order predicate calculus and thereby makes possible the definition of relational completeness and the rigorous study of good data base design.

The properties of relational data base management systems appear to be well suited to the ISS concepts of facilitating adaptation of software across systems to different requirements for data base manipulation and within a system to changing requirements. The separation of the concern for and specification of functional requirements from performance requirements is perhaps essential to the success of an ISS approach to system development. The data independence property of relational systems allows that separation.

The definition of a data base in terms of its natural structure only might allow the development of data as well as functional building blocks. The same data entities appear in several different $C^3$ systems. The data bases of these systems, however, may appear to be quite different, because they represent and structure the data differently, each reflecting the actual or anticipated accessing and performance requirements for the specific system. A data base definition reflecting a natural structure only and representing all information explicitly in terms of data values could provide an implementation independent model of the data for a given functional area. Such a model could be used directly in a relational data base system and as a source document for data bases to be developed for other kinds of systems. It would be desirable to gain some insight into the feasibility of developing a relational data building block for some $C^3$ functional area.

The separate specification of data base content and its machine representation and organization is consistent with the ISS concept of common functional requirements for which building blocks can be built without knowledge of system unique performance requirements and constraints. It first of all allows the developers of functional requirements and design to focus on the specification of the data base and operations on it before and without addressing performance and implementation issues. This separation and the resultant simplicity also permit the operational user, a key participant in the ISS concept, to address functional problems without knowledge of data processing issues.

Given a statement of data base operations, the data independence property of the relational data base model permits the tuning of a system's performance in light of actual workload and data accessing patterns, which may be difficult to predict prior to the utilization of a system or which may change during the system's life cycle. Requirements for this kind of adaptability of performance characteristics during the O&M phase of a system life cycle are among those identified for $C^3$ systems and targeted for ISS attention.

It is often the case that $C^3$ systems are proposed and developed to support operational concepts and environments that do not exist at the time that the system is being specified. It is therefore difficult to anticipate the kinds of data base operations that will be required to support the users in the performance of their tasks, although the data itself may be definable. Not only may there be no "normal" set of data base operations defined for the users but there will not be an appreciation of the data base manipulation requirements for different personal styles of interacting with a data base. Although research is being undertaken to develop a better understanding of user information requirements in $C^3$ systems, the burden is on the system to adapt to the vagaries of and evolution of user data manipulation requirements. The symmetry property of the relational data base model facilitates the support of different associations of entities in a data base, without redefinition of the data base or without performance penalties that can only be dealt with by restructuring of the data base.

The objectives of the application of a relational data base system are to develop an understanding of the advantages and disadvantages of the user interface properties of a relational system, and by necessity of a specific implementation, in the $C^3$ context. It is also an objective to gain some experience with the basic performance characteristics of relational data base systems and with the degree to which performance can be improved by adjusting the user-transparent machine representations and organizations to reflect the access patterns, workload, and performance requirements of a specific application. It is recognized that relational data base systems are not generally commercially available and are as yet untested in any large data-base system [MICH76]. It is also recognized that much work needs to be and continues to be directed at the achievement of greater efficiency in relational systems. It is believed, however, that in spite of the current impracticality of using a relational system in a $C^3$ acquisition program, the application and demonstration of a relational system will provide an opportunity to obtain first-hand experience with the use of an existing capability that has many of the characteristics deemed to be desirable for ISS building blocks.

## APPENDIX I

## MAN/MACHINE INTERFACE CHARACTERISTICS*

### Overall Characteristics of Position

I. User Role

A. Operator

1. Structured task and interaction with system

2. Forced pace

3. Monitoring or controlling through system outputs and inputs

4. Fast decision making -- initiate action through system within limited time constraints

5. Examples:

a. Air traffic control

b. Process control

c. Radar track monitoring

B. Analyst

1. Unstructured task and interaction with system

2. Self-paced

3. Data used as one input for decision making; may also rely on data from other sources, or experience, to recognize problem

4. Manipulation of data within system to establish relationships; use of on-line tools

---

* The man/machine interface characteristics in this appendix were developed and documented by Nancy C. Goodwin.

5. Result of decisions may or may not be entered into the system for direct action

6. Entry of decision may not control system directly

7. Decision may not be made immediately

8. Examples:

a. Intelligence analyst

b. Mission planning

c. Message handling

C. Data entry/service

1. User not source of data being entered

2. Retrieval in response to query; to answer specific, direct question

3. Data retrieval through highly structured sequences - user does not pull together information from several sources through multiple sequences

4. Structured task and interaction with system

5. Forced paced

6. Examples:

a. Airline reservations

b. Telephone directory assistence

c. Word processing center

II. Patterns of Use

A. Dedicated use

1. Terminal is located in primary workspace

2. During his shift, user has primary or sole access to terminal

3. Most of user's time is spent interacting with system

4. Most of user's job is accomplished through interaction with system

B. Non-dedicated use

1. Terminal is not in primary work area

2. Access to terminal casual or shared

3. User spends time on job away from terminal

4. User may not depend on system to accomplish many tasks

III. Data characteristics

A. Classified or not

B. Primarily graphic (tracks, drawings, status boards)

C. Primarily data tables

D. Primarily textual

IV. Training requirements

A. User available for training away from primary work space

B. Amount of time user can spend on training before accomplishing primary tasks on system

C. Amount of time user expects to spend with system after training (A vs. B in patterns of use above)

D. Transferability of previous skills to system use

V. Environment

A. Land, sea, air

1. Stability

2. Space requirements

B. Dedicated vs. shared space

1. Other demands on user's attention

2. Amount of space available for terminal/workstation

3. Competing requirements for light levels, noise levels, etc.

C. New facility vs. adapted facility vs. integration with existing facility

D. Number of users per terminal/workstation per shift

E. Multiple vs. specialized functions per terminal

## Workstation Characteristics - Physical Environment

I. Input devices

- A. Function keys
- B. Numeric keypad
- C. Alphabetic entry
  - 1. QWERTY keyboard
  - 2. Alphabetic order keyset

II. Output Devices

- A. Teletype
- B. CRT
  - 1. Graphic
  - 2. Alphanumeric
    - a. Primarily tabular data
    - b. Primarily textual data
- C. Status panels
- D. Transmission rate
- E. Full or half duplex
- F. Transmission mode
  - 1. Character
  - 2. Line
  - 3. Block
  - 4. Page

III. Control devices

A. Cursor positioning

1. Keys

2. Lightpen

3. Joystick

4. Trackball

5. Special (e.g., "mouse")

B. Display intensity controls

C. Focus

D. Audible alarm volume controls

IV. Console design

A. Desk/table size

B. Devices to be accommodated

C. Multiple purpose/dedicated purpose

D. Number of users per station

V. Room layout

A. Single/multi-purpose

B. Number of workstations

C. Number of users

D. Observers

E. Lighting requirements

## Software/Processing Requirements

## Directly Related to User Interaction

I. Interface Management -- manipulation of appearance and content of displayed data, which will not affect the content of the data base itself.

A. Control of display content -- specification by the user of the data to be displayed

1. Request data set

a. Request by id -- data set is preformatted and specified, and identified by name, number or other type of specific identification

i. id specified by typed entry

ii. id specified by menu selection -- displayed list of ids

iii. id specified by function key

b. Request subset -- user can specify portion of named set, by id

c. Request by characteristics - user specifies data set or subset wanted by listing characteristics of set, i.e., particular fields or values

2. Clear display -- remove data from the display without affecting data base

a. Entire display

b. Partial display

B. Control of display appearance -- given that a data set has been specified, the user can manipulate the appearance of the data on the display, or the amount of data on the display

1. Text-Tabular display

   a. Page -- the amount of data presented at one time on a display. If requested set exceeds display capacity, data are divided into multiple pages.

      i. Size

         a. Determined by terminal or system capacity

         b. Specified by user when making request

      ii. Display request -- how user accesses and changes pages

         a. Name or number entered

         b. Relative -- next or previous page requested

   b. Scrolling -- if requested dataset exceeds display capacity, the excess is accessed by line-by-line changes under user control

      i. Scroll up -- top line of data is deleted, next line added to bottom of display

      ii. Scroll down -- bottom line of data is deleted, earlier line added to top of display

      iii. Headers -- if tabular display, headers at top of columns identify data fields

         a. Stay on display during scrolling

         b. May scroll off display

   c. Formats -- describe the arrangement of the data on the display

      i. Size -- specifies how much data will appear

         a. Number of characters (text)

         b. Number of rows/columns (tab)

ii. Type -- describes number of items and associated values to be displayed

a. Summary - shows some or all data for multiple items

b. Individual -- shows some or all data for a single item.

iii. Selection -- associated with requesting data sets above

d. Clear -- deletion of data from the display

i. Delete entire display

ii. Delete part of display

2. Graphic displays -- consisting of images rather than text

a. Control of rate of presentation

i. Speed up

ii. Slow down

iii. Freeze

iv. Resume dynamic presentation

b. Control of orientation

i. Select view by name/id

ii. Recenter/offset

iii. Backup

iv. Zoom in

v. Zoom out

vi. Clear display

3. Display Coding -- technique used to call attention to, or distinguish among data items

a. Color

b. Blink

c. Highlight

d. Underline

C. Data functions -- various types of information may be presented to the user, whose control of the data is related to the function

1. Types of functions

a. System status -- number of users, system load, system in use, etc.

b. Error feedback -- notifies user that an input error has been made

c. Alarm/notice -- notifies user that system error has been made, emergency situation developed, input arrived requiring his attention

d. Command/menu -- lists of commands, files, displays, etc. that user may select as input to system

e. Read-only data -- user cannot edit directly on screen

f. Editable data -- user can change or edit these data directly, by making changes on display screen

2. Window usage -- areas available for two or more data types to be presented or used concurrently on one display

a. Size

i. Constant

ii. Varies depending on number of windows displayed

iii. Specified by user

b. Location

i. Constant

ii. Varies depending on number of windows displayed

c. Relationships/concurrence

i. Which windows can be displayed concurrently

ii. Which windows active concurrently

iii. Data moved from the window to another -- e.g., deleted from one, inserted in another

iv. Data copied from one window to another -- e.g., duplicated in both windows

v. Contents can be cleared independently of others

d. Window contents can be printed

e. Window contents can be stored in data base

f. Window contents can be transmitted

g. Window contents can be cleared

D. Presentation of feedback to user -- the user should receive feedback as a result of all inputs as well as status information. This may be presented on the terminal in form of audible alarms.

1. System status -- user should be aware of status of system elements which affect his interaction with the system.

a. System load (if heavy load implies slow response time)

b. Files available

c. Functions available

d. Position active

e. Date/time

2. Presentation of system status information

   a. On demand

   b. Periodically

   c. Constantly

3. Error feedback (See also Error Handling, Section III)

   a. Error noted in error message window

   b. Error message written in terms that user can understand

   c. Error message should be noticeable

4. Response to correct inputs -- all user inputs should be acknowledged in some way

   a. Notification if no data satisfy valid request

   b. Changes in data acknowledged, displayed, or highlighted

   c. Transmission of data acknowledged

   d. Notification if data exceeds display capacity so only partially displayed (i.e., page numbers given)

   e. Notification if data exceeds display capacity so none can be presented

5. System alerts or notices -- notify user if new data (or message) has arrived, if emergency notice has arrived, if dangerous situation is developing

   a. User may override or suppress alerts by category

      i. Routine

      ii. Previously seen

      iii. Specified category

b. User may remove alerts or attention-getters after having seen them (automatically vs. user action)

c. User may specify where (window, printer) alerts are sent

d. User may send alerts to other terminals

II. Data Base Manipulation -- functions needed to support the user's application, which may affect the content of the data base.

A. Text editing -- functions needed to support the entry of text and data into the data base

1. Type -- typed character appears on the display at cursor location, replacing character (if any) previously in that position; cursor moves right one space

2. Insert -- typed character appears on the display at cursor location; character (if any) previously in that location and cursor move right one space

3. Delete -- character marked by cursor is deleted from display; characters to right of deleted character move left one space. Cursor does not move

4. Move text -- marked string of characters is moved from one part of text to another

5. Copy text -- marked string copied

6. Delete text -- marked string deleted

7. Save text in named file -- text is saved and can be recalled explicitly by name, is saved beyond session end

8. Save text in named buffer -- text is saved and can be recalled explicitly by name, is not saved beyond session end

9. Save text in unnamed buffer -- text is saved temporarily, is recalled implicitly by buffer id or

command, is deleted by successive saves, is not saved beyond session end.

B. Graphic editing capabilities -- explicit changes to appearance of images on graphic display

1. Image creation -- specification of a new image on the display

2. Image deletion -- deletion of an image on the display

3. Update image -- change of an image on the display

C. Data manipulation/entry -- user can explicitly change content of data base during session

1. Entry -- addition of data to data base

2. Editing -- change value of data in data base

3. Deletion -- erase value from data base

D. User Control Functions

1. Cursor control -- cursor moved around display; character or image marked is not deleted by cursor

a. Lightpen

b. Keys

i. Up, down, right, left, home

ii. Word right, word left

iii. Line right, line left

2. Mode control -- user selects

a. Read only

b. Data entry

c. Graphics

i. "normal" -- system controlled

ii. Override

3. Command entry

a. Typed

b. Menu/list

c. Function keys

4. Item selection -- for update, tracking, detailed data display, deletion

5. User can interrupt sequence of actions if he does not want to continue. He will be returned to status before sequence begun.

E. Transmission -- user can send data to system, terminals, or printer

1. Data eligible for transmission

a. Only displayed data can be transmitted

b. Data <u>not</u> displayed can be transmitted

2. Transmit displayed dataset to:

a. External system

b. His stored files

c. Other users' files

d. Other users' terminals

e. Printer

3. User can transmit all or part of displayed data to 2a-e by specifying:

a. Name of displayed data

b. Area of display where data shown

c. All

4. Transmit named data not on display to 2a-e

5. Transmission control

   a. Data transmitted as result of explicit user action

   b. Data transmitted in response to request from system (e.g., status read periodically)

6. Feedback

   a. Data remains on display after transmission

   b. Data is cleared from display after transmission

   c. Transmission acknowledgement is printed or displayed

III. Error Handling -- consequences of user or system errors

A. User inputs will be checked before execution to ensure they are valid

B. Erroneous user inputs will not be erased from the display

C. User will be protected from making errors which destroy the database, whether files belong to him or to someone else

D. If error occurs partway through sequence of actions, user should not have to return to beginning of sequence - user should be able to correct error and continue

E. User will be notified if system errors occur

1. Type

2. Extent

3. Recovery techniques

IV. User-aids -- capabilities which are not necessary to accomplish functions, but which aid user in his interaction with system.

A. On-line help -- instructions describing job-oriented commands or sequences which can be accessed during job-oriented interaction

B. Pre-stored command sequences -- creation of "macro" commands to enable user to execute frequently used series of commands with single input

C. Guidance in error recovery -- user will be told type of error, place where error occurred (Section III, Error Handling)

D. Automatic update of related files -- single data entry results in multiple updates of that item when item appears in multiple files

E. Automatic form filling -- system fills in data entry form as much as possible when data is available in data base - user is saved from entering data already in system

F. Alternate data entry techniques -- user given choice of data/command entry techniques, may select one which suits his personal style, preferences, or experience

## Software/Processing Requirements

### Indirectly Related to User Interaction

I. Performance characteristics -- system ability to support multiple users, multiple functions, with adequate response time.

A. Capacity -- Terminals

1. Number of user terminals that system will handle

2. Number of different functional roles terminal can handle

3. Relationship of terminals -- are they independent or slaves? Do they need to communicate with each other?

B. Capacity -- Users

1. Number of users -- individuals -- that system can handle

2. Number of different functional roles system can handle

C. Capacity -- Data Base

1. Number of data files to be supported

2. Rate of data input to system from non-user sources

3. Volume of data input from non-user sources

4. Rate of data input from users

5. Volume input from users

6. Rate output from system

7. Volume output from system

D. Response time -- how quickly does system respond to user inputs (commands, instructions, or function keys)

1. Terminal level -- return of cursor, echo of typed character

2. System level -- execution of command correctly results in a noticeable feedback within specified time(s) (may vary according to complexity of command function)

3. Execution of command incorrectly results in error feedback within specified time

II. System Monitoring/Control Functions

A. Access Controls

1. Files

a. Read

b. Write

2. Functions - who can do what

B. Routing Controls

1. System to user/terminal

2. User to system

3. User to user

C. Maintenance

D. Data Collection

E. System Performance Monitoring

1. Response time

2. Load effects

## REFERENCES


AIRF75 AF Regulation 800-14, Volume II, Acquisition and Support Procedures for Computer Resources in Systems, 26 September 1975.

ALLE76 F. Allen, J. Cocke, "A Program Data Flow Analysis Procedure," CACM Vol. 19, No. 3, March 1976, 137-147.

AUER "Auerbach Computer Technology Report," Auerbach Publishers, Inc.

BOEH77 B. Boehm, R. McClean, D. Urfrig, "Some Experience with Automated Aids to the Design of Large-Scale Reliable Software, Proceedings of the International Conference on Reliable Software, April 1975, 105-113.

BRIN77 P. Brinch Hansen, "Experience with Modular Concurrent Programming," IEEE Transactions on Software Engineering, Vol. SE-3, No. 2, March 1977, pp. 156-159.

CANN75 R. Canning, "Progress Toward Easier Programming," EDP Analyzer, Vol. 13, No. 9, September 1975.

CARL76 W. Carlson, "Software Research in the Department of Defense," Proceedings, 2nd International Conference on Software Engineering, 1976, pp. 379-383.

CHAM76 D. D. Chamberlin, "Relational Data Base Management Systems," ACM Computing Surveys, Vol. 8, No. 1, March 1976, pp.43-66.

CODD70 E. F. Codd, "A Relational Model of Data for Large Shared Data Banks," CACM, Vol. 13, No. 6, June 1970, pp. 377-387.

COOP75 Cdr. J. Cooper, "Increased Software Transferability Dependent on Standardization Efforts," Defense Management Journal, October 1975.

ENGE76 J. Engelland et al., Operational Software Concept (Phase Two), AFAL TR-75-230, Air Force Avionics Laboratory, Wright-Patterson AFB, Ohio, January 1976(AD A021 327).

FOSD76 L. Fosdick, L. Osterweil, "Data Flow Analysis in Software Reliability," Computing Surveys, Vol. 8, No. 3, September 1976, pp. 305-330.

GLOR77 J. Glore, "Software Acquisition Management Guidebook: Life Cycle Events," ESD TR-77-22, Electronic Systems Division, USAF, Bedford, MA, February 1977.

GORD73 E. Gord, M. Hopwood, "Nonhierarchical Process Structure in a Decentralized Computing Environment, Technical Report #32, Department of Information and Computer Science, University of California, Irvine, CA, June 1973.

HAMM74 M. Hammer, W. G. Howe, I. Wladawsky, "An Interactive Business Definition System," SIGPLAN Notices, Vol. 9, No. 4, pp. 25-33, April 1974.

HAMM75 M. Hammer, W. G. Howe, V. Kruskal, I. Wladawsky, "A Very High Level Programming Language for Data Processing Applications," IBM Research Report RC5583, August 15, 1975. (to be published)

JACK77 K. Jackson, "Language Design for Modular Software Construction," Information Processing 77, B. Gilchrist, ed., North-Holland Publishing Company, Amsterdam, 1977, pp. 577-581.

MICH76 A. S. Michaels, B. Mittman, C. R. Carlson, "A Comparison of Relational and CODASYL Approaches to Data-Base Management," ACM Computing Surveys, Vol. 8, No. 1, March 1976, pp. 125-151.

MIX77 M. Mix, et al., Standard Software Base, RADC-TR-77-99, Rome Air Development Center, Griffiss AFB, NY, March 1977.

NELC76 Advanced Command and Control Architectural Testbed (ACCAT) Program Management Plan FY1977, Volume I - Management Plan, Prepared by NELC for ARPA/NAVELEX03, 15 November 1976.

NYMA77 T. Nyman, DoD Software Research and Development Technology Program Plan," Abridged Proceedings from the Software Management Conference Series 1977, AIAA, Los Angeles, 1977, p. 58.

ROWE75 L. Rowe, The Distributed Computing Operating System, TR #66, Department of Information and Computer Science, University of California, Irvine, CA, June 1975.

RUMB77 J. Rumbaugh, "A Data Flow Multiprocessor," IEEE Transactions on Computers, Vol. C-26, No. 2, February 1977, pp. 138-146.

RUTH76 G. R. Ruth, "Protosystem I: An Automatic Programming System Prototype," AD 026912, Office of Naval Research, Arlington, VA, July 1976.

SMIT76 R. Smith, System Partitioning Study Final Report, MDC G6603, McDonnel Douglas Astronautics Company-West, Huntington Beach, CA, December 1976.

SULL73 J. Sullivan, "Egalitarian Programming," unpublished project note, August 1973.